Name: Achyut Shukla Batch: CS – ML – 1 PRN: 20070122005

॥वसुधैव कुटुम्बकम्॥

SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY)

(Established under section 3 of the UGC Act 1956)

Re - accredited by NAAC with 'A' Grade

Founder: Prof.Dr. S. B. Mujumdar, M.Sc.,Ph.D. (Awarded Padma Bhushan and Padma Shri by President of India)

Lab Assignment – 3

Aim:

Apply the K-Nearest Neighbours Classifier algorithm on a sample case study and data set. Evaluate Results.

PART – A

Theory:

1. **Explain working of KNN:** K-Nearest neighbour's (KNN) is a simple and effective machine learning algorithm used for classification and regression tasks. The algorithm is based on the idea that data points with similar features tend to belong to the same class or have similar output values. KNN is a non-parametric and lazy learning algorithm, which means it doesn't make any assumptions about the underlying data distribution and doesn't learn an explicit model during training.

   Here's how the KNN algorithm works:

   i. **Data Preparation**: First, you need a labelled dataset containing input features and corresponding output labels (for classification) or output values (for regression). Each data point in the dataset is represented by a vector of features. For example, in a binary classification task, each data point could have two features (x1, x2) and a label (0 or 1).

   ii. **Choosing the Value of K**: The "K" in KNN represents the number of nearest neighbours that will be considered to make predictions. It's a hyperparameter that you need to set before training the algorithm. A small value of K may lead to noisy predictions, while a large value may cause the algorithm to overlook local patterns. Choosing an appropriate value for K is essential and often requires experimentation.

   iii. **Calculating Distance**: KNN relies on measuring the distance between data points in the feature space to identify the nearest neighbours. The most used distance metric is Euclidean distance, but other distance metrics like Manhattan distance or Makowski distance can also be used depending on the nature of the data. Euclidean distance between two points (p1 and p2) in a 2D feature space is given by:

distance = sqrt ((p1.x - p2.x) ^2 + (p1.y - p2.y) ^2)

iv. **Finding K Nearest Neighbours**: For a given data point that we want to classify or predict, the algorithm calculates the distance to all other data points in the training dataset. It then selects the K data points with the smallest distances (i.e., the K nearest neighbour's).

v. **Majority Voting (Classification) or Averaging (Regression)**: For classification tasks, KNN takes the majority vote of the K nearest neighbour's classes and assigns that class to the data point in question. In regression tasks, KNN takes the average of the output values of the K nearest neighbour's and uses it as the predicted output for the data point.

vi. **Prediction**: After determining the class (for classification) or value (for regression), the algorithm assigns the class or value as the prediction for the given data point.

vii. **Evaluation**: Once the predictions are made for all data points in the test set, the performance of the KNN algorithm is evaluated using appropriate metrics such as accuracy (for classification) or mean squared error (for regression).

It's important to note that KNN is a memory-intensive algorithm, as it needs to store the entire training dataset to make predictions. Additionally, the choice of distance metric and the value of K can significantly impact the algorithm's performance, so careful consideration and tuning are necessary.

2. **What is Distance weighted KNN?**

Distance-weighted KNN (DWKNN), also known as weighted KNN, is a variant of the K-Nearest neighbour's (KNN) algorithm. While the traditional KNN assigns equal weights to all K nearest neighbour's when making predictions, DWKNN considers the distances between the data point to be predicted and its neighbour's and assigns weights to each neighbour based on these distances.

In standard KNN, all K nearest neighbours have an equal say in the final prediction. However, this approach may not be optimal when some neighbours are significantly closer to the query point than others. DWKNN addresses this issue by giving more weight to the closer neighbour's and less weight to the ones that are farther away. The idea is that closer neighbours are likely to have a more significant influence on the prediction since they are more like the query point.

The weight assigned to each neighbour can be calculated using various functions. One common method is to use the inverse of the distance, i.e., the closer the neighbour, the higher the weight. The weight (w_i) for the i^{th} neighbour can be calculated as follows:

W_i = 1 / distance (query point, neighbour)

Once the weights are assigned to each neighbour, the algorithm performs a weighted majority vote for classification tasks or a weighted average for regression tasks. For classification, the class with the highest cumulative weight among the K neighbour's is assigned as the predicted class. For regression, the output value is calculated as the weighted average of the output values of the K nearest neighbours.

The use of distance weighted KNN can improve the performance of the algorithm, especially when the data distribution is uneven or when certain neighbours are more reliable predictors than others. However, determining the appropriate weight function and tuning its parameters can be challenging and may require experimentation.

3. **Strengths and Weaknesses of KNN**

K-Nearest neighbour's (KNN) is a popular and simple machine learning algorithm with its own set of strengths and weaknesses. Understanding these aspects can help in choosing the right situations to apply KNN and be aware of its limitations:

Strengths:

a) **Simple and Intuitive:** KNN is straightforward to understand and implement, making it an excellent starting point for newcomers to machine learning and classification problems.

b) **No Training Phase:** Unlike many other algorithms, KNN does not have an explicit training phase. It memorizes the training data, and the prediction step involves only finding the nearest neighbours.

c) **Non-Parametric:** KNN makes no assumptions about the underlying data distribution, making it suitable for both linear and non-linear data.

d) **Flexibility with Data:** KNN can handle multi-class classification tasks, binary classification, and regression (with some adaptations). It can also handle data with complex relationships and interactions.

e) **Adaptable to New Data:** As the algorithm doesn't build an explicit model during training, it can adapt to new data easily, making it useful for online or incremental learning scenarios.

f) **Robust to Noise:** KNN can still provide reasonable predictions even when the data contains noise or outliers because it considers multiple neighbours.

Weaknesses:

a) **Computationally Expensive:** As KNN needs to find the nearest neighbours for each data point, the computational cost increases with the size of the dataset. It can be impractical for large datasets.

b) **High Memory Usage:** KNN requires storing the entire dataset, which can be memory-intensive for large datasets. This can limit its use on memory-constrained systems.

c) **Sensitive to Irrelevant Features:** The algorithm treats all features equally, so irrelevant features can negatively impact performance. Feature scaling and selection are essential preprocessing steps.

d) **Hyperparameter Sensitivity:** The choice of K (number of neighbours) significantly affects the algorithm's performance. Selecting the right value of K requires careful experimentation.

e) **Not Suitable for High-Dimensional Data:** In high-dimensional feature spaces, the notion of distance becomes less meaningful (known as the "curse of dimensionality"), and KNN may struggle to find meaningful neighbours.

f) **Imbalanced Data Handling:** KNN may Favor the majority class in imbalanced datasets due to the majority of neighbours coming from the dominant class. Special techniques like weighted KNN can help address this.

g) **Distance Metric Selection:** The choice of distance metric can impact the results significantly. Using an inappropriate metric can lead to suboptimal performance.

PART – B

Experiment:

Step 1 – Study of Dataset (Dataset references are given below)

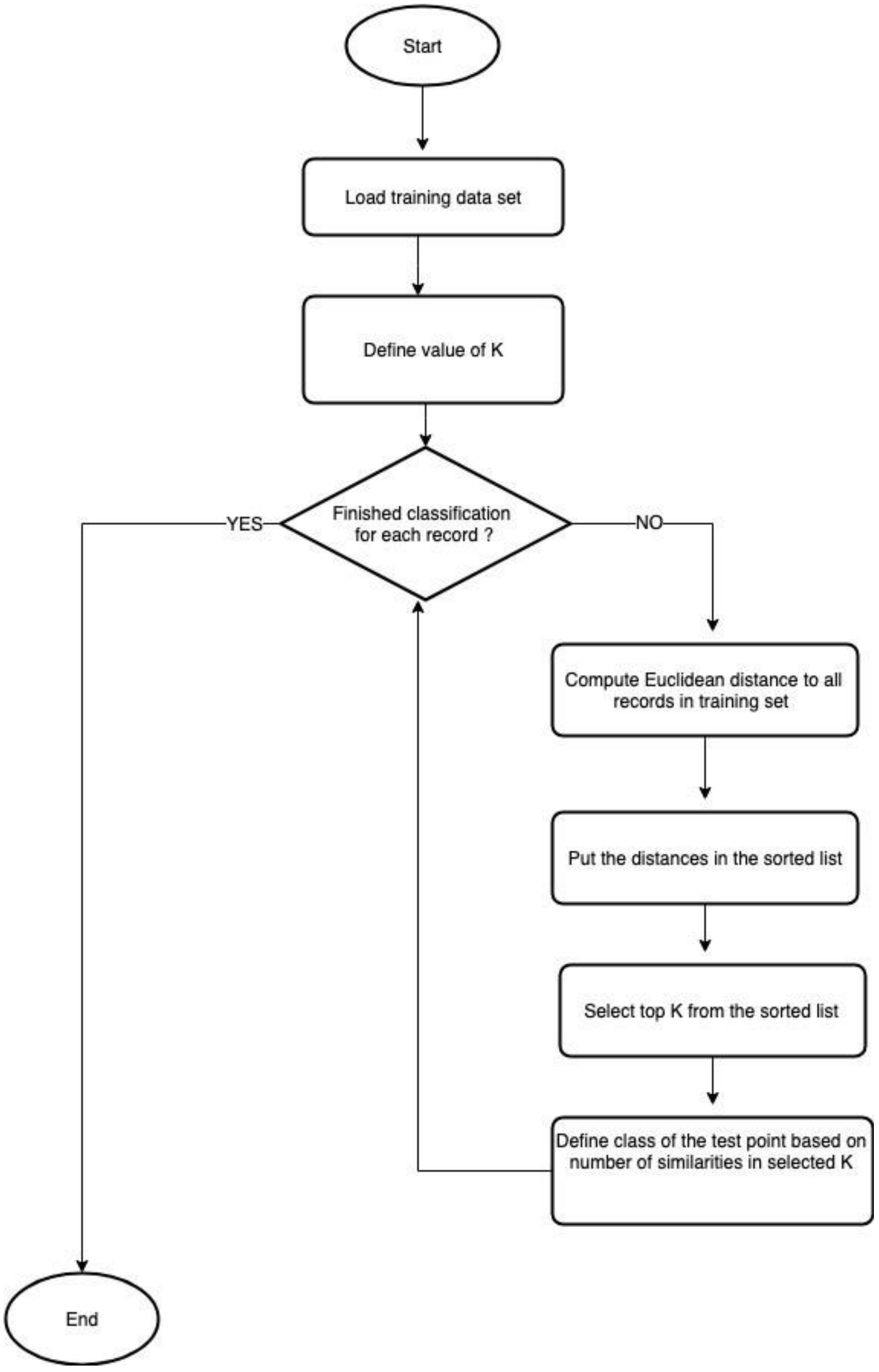
Step 2 − choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 − For each point in the test data do the following −

- 3.1 − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- 3.2 − Now, based on the distance value, sort them in ascending order.
- 3.3 − Next, it will choose the top K rows from the sorted array.
- 3.4 − Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 – End

Name: Achyut Shukla Batch: CS – ML – 1 PRN: 20070122005


Start
Load training data set
Define value of K
Finished classification for each record ?
YES
NO
Compute Euclidean distance to all records in training set
Put the distances in the sorted list
Select top K from the sorted list
Define class of the test point based on number of similarities in selected K
End

Datasets :

1. https://www.kaggle.com/datasets/rakeshrau/social-network-ads

Name: Achyut Shukla Batch: CS – ML – 1 PRN: 20070122005

2. https://www.kaggle.com/datasets/devansodariya/student-performance-data

Output:

1. Data Visualisation
2. Graphs

Inference Discussion